LA-11769-MS

UC-905
Issued: February 1990

LA--11769-MS

DE90 006425

Performance Characterization of the Convex C-240 Computer System

Rebecca J. Koskela
Margaret L. Simmons
Harvey J. Wasserman

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

PERFORMANCE CHARACTERIZATION OF THE CONVEX C-240 COMPUTER SYSTEM

by

REBECCA J. KOSKELA, MARGARET L. SIMMONS, and HARVEY J. WASSERMAN

ABSTRACT

This paper considers the sequential, vector, and parallel processing performance of the Convex C-240 computer system. A set of well-characterized Fortran benchmarks that have been run on a wide variety of computer systems was used for the study. The data from the C-240 are compared with those obtained on a CRAY X-MP system as well as with the Multiflow Trace-14/300 system The results suggest that the C-240 system is very nearly capable of providing throughput power equivalent to that of the CRAY X-MP/14 we tested.

1. Introduction

We recently reported a comprehensive performance evaluation of three computer systems that have come to be known as minisupercomputers.[1] Among the machines tested was the Convex C-1/XP, a single-processor machine with a clock cycle of 100 nanoseconds (ns). In this paper we characterize the performance of a four-processor successor to the C-1/XP, the Convex C-240. The evaluation considers the single-processor performance in a dedicated environment as well as two aspects of the concurrent nature of the C-240. Of the manufacturers mentioned in our original paper, Convex is the only to produce a second-generation machine.

Our interest in benchmarking computer systems extends back many years.[2-7] The emphasis has largely been on supercomputer architectures because of the challenge presented by the heterogeneous nature of these machines. Supercomputers may be described as machines whose hardware consists of certain advanced architectural features implemented with state-of-the-art device technology.[1] The architectural feature that most distinguishes supercomputers is parallelism, both in the CPU (examples are multiple CPUs; vector processors; chaining; and multiple, independent, pipelined functional units) and in the memory (examples are multiply banked main memory, local memory, vector registers, and solid-state external storage devices). Supercomputer hardware must also be accompanied by software, particularly a compiler, that can extract parallelism from a program and make it available to the hardware.

The Convex computers, and other machines like them, are also of interest to us because their architectures contain many of the same features that contribute to supercomputer performance. Thus, we define the term *minisupercomputer* to mean machines whose architectures feature many, if not all, of the characteristics of supercomputers, but whose device technology uses less expensive and more readily available components.[1] Note that these definitions of *supercomputer* and *minisupercomputer* are independent of system price, observed performance, and expected user class or "market niche."

This benchmark of the Convex C-240 used the Los Alamos Benchmark set that consists of thirteen Fortran codes representative of the workload of the Laboratory. These codes have been described elsewhere.[2] In addition, we ran four codes that have been explicitly partitioned for parallel processing. A recent publication has appeared describing in detail the parallel performance of the C-240 on two of our codes.[8]

2. C-240 Overview

A detailed description of the Convex C-240 architecture may be found elsewhere.[9] A C-240 consists of four processors connected to a maximum of 1024 Mbytes of complementary metal oxide semiconductor (CMOS) memory. Our benchmark was of a machine with 512 Mbytes interleaved 32 ways. Memory is implemented on 120-ns dynamic RAM (DRAM) chips. The clock cycle of the machine is 40 ns. The vector functional units, however, have a cycle time of 80 ns, but because there are two pipes, the effective time per result can be 40 ns. There is a 4-kbyte scalar data cache as well as an 8-kbyte instruction cache.

The software we used included the Convex UNIX OS (4.3BSD) Version 7.0, the Convex Fortran compiler, fc 5.0, and associated support libraries. The Convex Fortran compiler attempts to automatically parallelize code at the DO-loop level if there are no dependences between iterations of the loop in question. As with any vectorizing compiler, the first attempt is to vectorize inner loops and then parallelize outer loops, interchanging loops if necessary for best performance. There are many command-line options to assist the compiler in its attempts to both vectorize and parallelize code. In addition, there are inline directives to further assist optimization during compilation.

3. Results

3.1 Single-Processor Tests

Table 1 shows the results of the single-processor tests. The C-240 times show considerable improvement over the C-1/XP times of our last benchmark,[1] ranging from about twice as fast to almost six times as fast. The C-1/XP had basically the same architecture as the C-240 with a slower clock period. Improvements in Convex software probably account for most of the C-240's performance improvement above what the decreased clock cycle would provide.

Again, these are single-processor times for the Convex. When we used the compiler option that directs the compiler to attempt parallelization, only WAVE showed any improvement in runtime. The four-processor time for WAVE was 323.0 seconds. This speedup results from concurrent vector (CV) computation in which different iterations of vectorizable loops are computed on different CPUs in parallel.

There are two possible reasons why the other codes did not improve with the compiler option allowing CV computation. One is that it is difficult for the compiler to discover the parallelism present because of the way the codes are structured. Subroutine and function calls in the loops are often the culprits in this regard. Another reason is that, for a given loop, CV computation reduces the vector length that the hardware executes, and the relatively short loop lengths in our codes mean inefficient use of the vector hardware. The C-240 has a maximum vector register length of 128 elements. In WAVE, the predominant vector length is 256, whereas in the other codes the vector length does not exceed 100. Both of these factors—the difficulty in extracting parallelism and the shorter hardware vector lengths resulting from CV computation—were also found in another parallel minisupercomputer we benchmarked, the Alliant FX system.[1]

Table 1. Comparison of Single-Processor Benchmark Execution Times[a]

Code	*Convex C-1/XP Time*	*Convex C-240 Time*	*CRAY X-MP/14 Time*	*Ratio: C-240 / X-MP*	*Multiflow Trace -14/300 Time*
FFT	97.0	14.7	3.7	4.0	17.5
GAMTEB	61.0	10.7	5.6	1.9	12.1
SCALGAM	446.4	242.1	82.7	2.9	403.5
PHOTON	-	338.2	120.3	2.8	553.5
LSS	65.0	23.5	6.3	3.7	20.7
MATRIX	529.0	140.0	34.9	3.8	259.0
INTMC	83.0	31.1	13.0	2.4	26.8
HYDRO	298.2	74.6	24.7	3.0	206.7
WAVE	-	410.2	169.9	2.4	816.0
ESN	-	41.9	17.2	2.4	56.7
VGAM [b]	-	4.4	1.6	2.8	16.4
MCNP [c]	-	2525.9	940.5	2.7	-

(a) Times are in seconds.
(b) 40,000 source particles.
(c) 60,000 source particles.

For comparison, we have included in Table 1 benchmark times from a CRAY X-MP/14, which was the only Cray Research computer at the Laboratory running UNICOS at the time this study was carried out. (This is a 9.5-ns machine running CFT77 3.0.2.) A single processor of the C-240 provides between one-quarter and one-half the performance of this single-processor X-MP on all of our benchmark codes. However, there are three codes in the benchmark suite that more closely resemble production codes at the Laboratory, and we tend to place more emphasis on their performance. They are HYDRO (ratio C-240/X-MP = 3.0), WAVE (2.4), and MCNP (2.7).

It is important to note the effect of the compilers on HYDRO. With CFT 1.16, the times for the X-MP/14 and the (single-processor) C-240 are nearly equal. This is because of several critical loops that do not vectorize with CFT 1.16 but do vectorize using both CFT77 and the Convex C-240 compiler. Most of the loops are relatively large (on the order of 50 statements) and contain many conditionals.[10]

Note that the ratio of the clock cycle of the X-MP to that of the C-240 is about 4.2. On all of our benchmarks, the performance of the C-240 relative to the X-MP is better than one would expect based on only the clock ratio. Although not shown in Table 1, we have calculated the ratios of the single-processor C-240 times to the times from a single processor of the CRAY Y-MP/864 (Serial Number 1010, running UNICOS and CFT77 3.0). We computed these ratios to show the performance that a single C-240 processor provides relative to the full range of Cray processors. The C-240-to-Y-MP ratios (again, both single-processor) are roughly in the range 3–7; here the ratio of the CPU clocks is 6.7.

Table 1 also lists execution times for another minisupercomputer we benchmarked recently, the Multiflow Trace-14/300.[11] The Trace is a VLIW (very long instruction word) machine with a 130-ns cycle time. Comparing the execution times for the Trace and the C-240, we see that the Multiflow runs faster in two cases. One of these is INTMC, a benchmark with integer computation only. We'll discuss the other of these, LSS, in a moment. We also see that the performance of the two machines is closest on FFT, GAMTEB, SCALGAM/PHOTON, and ESN. All these codes are scalar except for FFT, which involves short vector lengths (in the range 2–64). Vector lengths in this range are especially short for the C-240. The ratios of execution times for these four codes on the two minisupercomputers are in the range 1.1–1.7. On the vector codes in the benchmark suite, however, the performance gap between the two machines widens further (ratios in parentheses): HYDRO (2.8), WAVE (2.5), MATRIX (1.8), and VGAM (3.7). The only exception is LSS, the one code on which the Multiflow performs better than the Convex. On LSS, the Multiflow inlines SAXPY and unrolls th SAXPY loop to a large depth. We tried to inline SAXPY on the Convex, but the compiler refused to do the inline because of the adjustable arrays in SAXPY. Curiously, compiling LSS for vector concurrent mode on the C-240 failed to yield more encouraging results, possibly as a result of the short vector lengths that occur in SAXPY.

3.2 Basic Vector Operations

Rates, in millions of floating-point operations per second (MFLOPS), for simple vector operations on a single processor of the C-240 are shown in Table 2. As is well known, these tests measure the time to do one million vector operations. The one million operations are distributed in two loops: an inner loop that runs over the vector length of interest and an outer loop that just repeats the inner loop enough times to get a measurable time. Some compilers realize that there is redundant work involved in these two loops and optimize them in various ways that defeat the purpose of the tests. The Convex compiler is a case in point.

Table 2. Convex C-240 (Single Processor) Rates (in MFLOPS) for Selected Vector Operations as a Function of Vector Length

Operation	*10*	*50*	*100*	*200*	*1000*
V=V+S	4	9	11	11	12
V=V+S(i=1,n,23)	2	9	11	11	11
V=V+S(i=1,n,8)	2	6	6	6	6
V=V*V	3	7	7	8	8
V=V+S*V	6	13	15	15	15
V=V*V+V*V	6	13	14	14	14
V(J(I))=V*V	2	4	5	5	5

The rates that we first observed for these tests were roughly twice those shown in Table 2. However, some inconsistencies in the data prompted us to look at the code the compiler generated for a sample loop. In doing so we found that the compiler had performed a "hoist and sink" optimization in which it generated a load for one set of vector operands, iterated on only the floating point instruction, and then did a single store. This particular type of optimization, which was also seen in the Fujitsu VP-200 benchmark,[12] is more insidious than other optimizations that subvert this benchmark. Other optimizations, such as those performed by Cray CFT77, simply remove *all* instructions from the loops. This is easy to spot even though the code doesn't check the answers, because the observed rates are unreasonably large (~ gigaFLOPS range). The observed rates for the C-240, on the other hand, were reasonable. This demonstrates the increasing sophistication of optimizing compilers and the need to validate that the intent of certain benchmarks is being maintained.

The solution to the hoisting problem on the C-240 was to add a subroutine call using the vector operands as arguments in the outer loop. The overhead for the subroutine call was then subtracted from the observed time for each operation, and the results are shown in Table 2.

The C-240, like its predecessor, has only a single port to memory. The effect of this can be seen in Table 2. For example, the CRAY Y-MP, with two read ports and one write port, can fully support the bandwidth required for the (one read, one write) operation V = V + S. As such, the Y-MP performs at about 90% of its peak rate on this operation. The C-240, however, really has only one-half the bandwidth required for V = V + S, and we observe a rate (at vector length 1000) of about half the peak rate for this operation. Of course, the number of ports to memory is only one of many features that contribute to performance. For example, the CRAY-2 also has one port. A single processor of the CRAY-2S achieves about 35% of its peak rate on this operation, because the memory is relatively slow in comparison with the CPU speed.[13] The bandwidth per port on the C-240 is about 200 Mbytes per second.

The observed decrease in rate between the operations V = V + S and V = V * V is also due to the single port.

The time data as a function of vector length for each operation may be least-squares fit using a three parameter model that gives the vector startup time, the time per result, and the "stripmine" time which is the time to reload the vector registers after each 128-element vector has been processed. A plot of time per element vs vector length, showing both the experimental data (as Xs) and the least-squares fit for the operation V = V + S, is shown in Figure 1. The fit yields a time per result of nearly 80 ns. In other words, on average, the C-240 can return a result only every two clocks although it can overlap the vector load with the multiply, it must wait until the load is finished before doing the store, because there is only one port.

We also measured the rates for simple vector operations as a function of vector length and vector stride for strides 2, 4, 8, 23, and 49. At vector length 1000 only stride 8 (third entry in Table 2) shows an appreciable degradation in rate (about 50% of the contiguous case). Because the memory is interleaved 32 ways, with stride 8 we access only four unique banks. The other strides show small degradations at shorter vector lengths because of additional startup time.

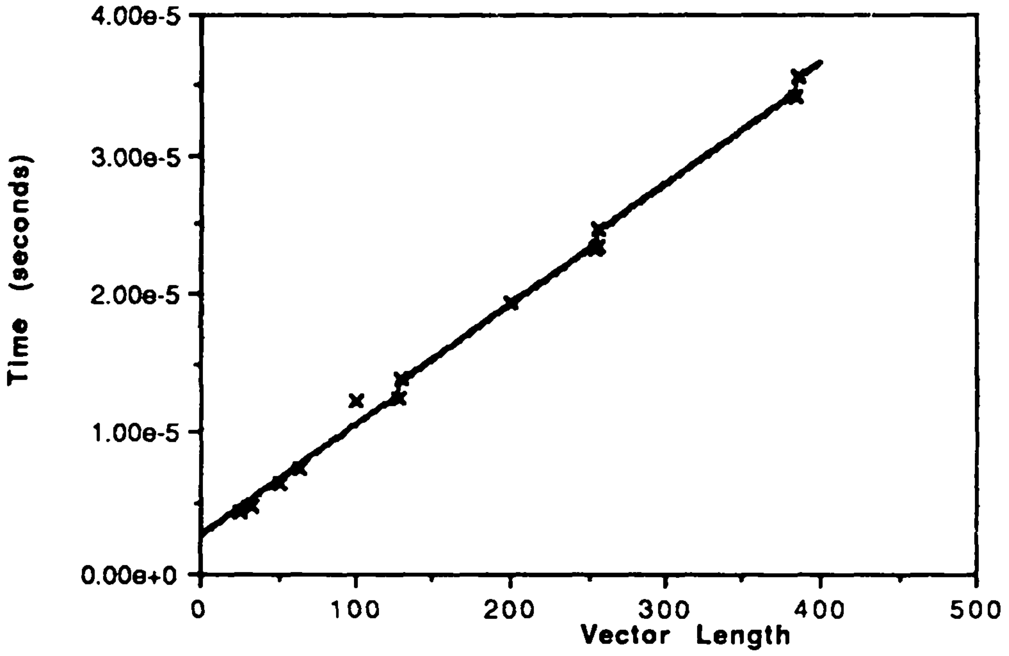

Figure 1. Least-squares fit of time data vs vector length for the operation V = V + S on the Convex C-240. Experimental data are indicated by Xs. Parameters derived from the fit are startup time = 1922.0 ns, element time = 78.1 ns, and stripmine time = 841.4 ns.

4. Multiprocessing

In a recent CRAY Y-MP benchmark[14] we considered multiprocessing performance in two different ways. The first was to use several benchmark codes explicitly coded for multitasking and measure the decrease in turn-around time for a single job. The second way was to measure the *degradation* in processing rate that occurred when a code was timed while a specific "load" ran in the other processors. This second test is designed to estimate how benchmarks will perform under more realistic user conditions as opposed to the dedicated runs described above. Note, however, that these data are often indications of worst-case behavior.

We decided to carry out both of these types of tests on the Convex C-240. We discuss the multitasking results first.

4.1 Multitasking

Elapsed (wallclock) times for four parallel benchmarks are listed in Table 3. A plot of speedup for the codes is shown in Figure 2. The speedup, S, for n processors, is given by

$$S = T(serial) \; / \; T(n) \; ,$$

where T represents time. The parallel code HYDRO is not listed in the table because the Convex parallel processing system does not include event synchronization, which is currently heavily used by HYDRO.

Table 3. Execution Times (seconds) Using Multiple Processors on the Convex C-240

Code	*Serial Time*	*1 Proc.*	*2 Proc.*	*3 Proc.*	*4 Proc.*
ESN	42.0	45.6	24.8	18.2	14.1
GAMTEB	10.7	14.5	14.6	7.7	5.5
SCALGAM	242.1	377.3	187.7	:26.1	96.0
MCNP (60,000 sources)	2393.6	2525.9	1212.9	813.1	660.8

Parallel processing on the Convex C-240 is intended to be automatically handled by the compiler. However, the four codes shown in Table 3 were explicitly partitioned to run in parallel. This approach, plus the use of some directives, was necessary for the compiler to recognize the parallelism in the codes. This type of restructuring has been necessary on all multiprocessors on which we have run.

The speedups on the four-processor C-240 range from 1.95 on GAMTEB to 3.6 on MCNP. The speedup on MCNP compares favorably with the measured speedup of 3.65 on the CRAY X-MP/416.[13]

It is difficult to explain why speedups on the other codes are not as large as we might expect. We know that the granularity of the tasks in GAMTEB and SCALGAM is smaller than that of MCNP, despite the overall similarity in the algorithms (Monte Carlo particle transport). Notice that the overhead for multiprocessing, which is estimated by taking the ratio of the one-processor time to the sequential time, is large on SCALGAM and GAMTEB. For MCNP it is in line with what it is on the CRAY-2 and CRAY X-MP/416. The small task granularity in GAMTEB also accounts for the one-processor and two-processor times being the same. GAMTEB contains a loop that spawns NPROC – 1 processes, where NPROC is the number of processors. The main code then runs the multitasked routine as well. When NPROC is two, the main code processes all particles before the child task even starts. It is not at all clear, however, why the overhead associated with SCALGAM is so large or why the parallel efficiency of ESN, which involves no explicit synchronization, is only 75%. The parallel benchmark codes are still in the development stage, however, and we plan to produce a more robust set soon.

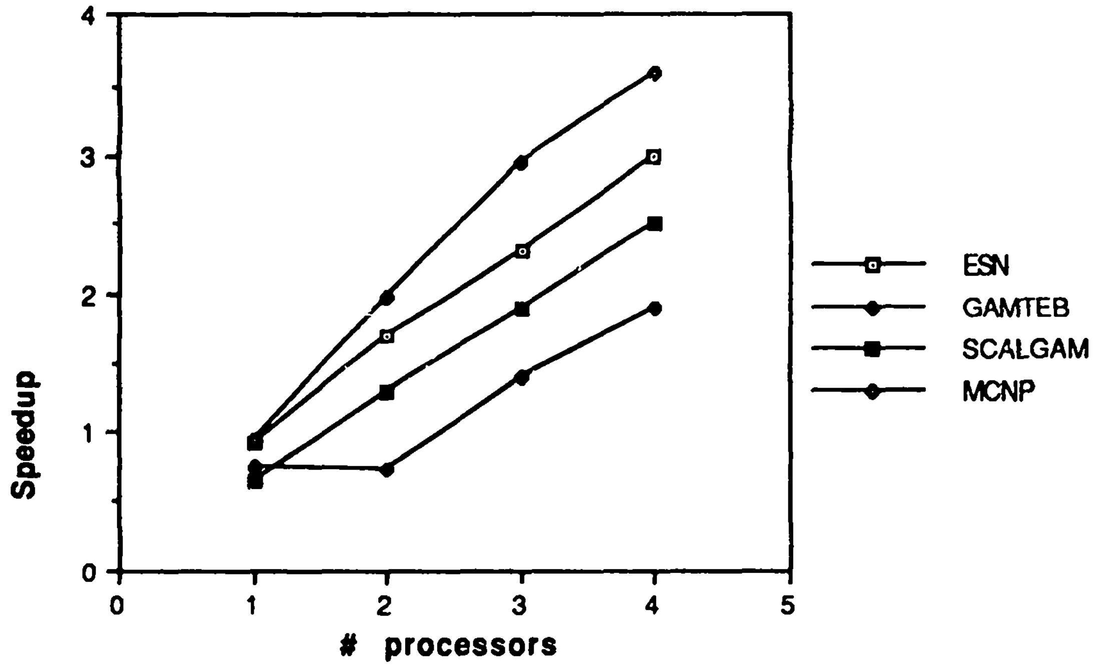

Figure 2. Multiprocessing speedup as a function of processing power on the Convex C-240.

4.2 Memory Contention Studies

We now consider the performance of some benchmarks under loaded conditions to test the effect of memory conflicts. Tables 4a–4e show the results of these tests. Each table lists data for a specific "probe" code, which is what was timed. In Tables 4a–4d, the probes are simple vector operations, either V = V + S or V = V + S * V (SAXPY), for which we measured the rate in MFLOPS. The probes were measured at vector length 1000.

Table 4a. Performance of V = V + S in MFLOPS under Various Loads

	Number of Processors Running the Load Code			
Load Code	*None*	*1*	*2*	*3*
V = V + S	12.0	11.5	11.0	10.5
V = V + S * V	12.0	11.5	11.0	10.5

Table 4b. Performance of V = V + S * V in MFLOPS under Various Loads

	Number of Processors Running the Load Code			
Load Code	*None*	*1*	*2*	*3*
V = V + S	15.0	15.0	15.0	14.0
V = V + S * V	15.0	15.0	15.0	15.0

Table 4c. Performance of V = V + S in MFLOPS under a Strided Load

	Number of Processors Running the Load Code			
Load Code	*None*	*1*	2	*3*
V = V + S (stride 8)	12.0	5.5	3.2	2.8

Table 4d. Performance of V = V + S * V in MFLOPS under a Strided Load

	Number of Processors Running the Load Code			
Load Code	*None*	*1*	2	*3*
V = V + S (stride 8)	15.0	7.3	4.6	3.6

Table 4e. Execution Time for HYDRO (seconds) while Running HYDRO as Load

	Number of Processors Running the Load Code			
Load Code	*None*	*1*	2	*3*
HYDRO	74.6	79.5	85.9	92.6

For these operations the "load" codes, the codes running in the processors besides the one on which the probe is running, are also simple vector operations: either V = V + S at stride 1, V = V + S * V at stride 1, or V = V + S at stride 8, again all at vector length 1000.

The data show that no appreciable degradation in rate occurs for either probe operation when the load operation accesses memory contiguously. This is not what was observed on the CRAY X-MP/416, where, for example, four copies of SAXPY ran at about 70% efficiency.[15] We believe the C-240 shows less degradation than the X-MP for the following reason: The memory bottleneck occurs at the processor level, not at the bank level, because there is only a single port to memory. That is, there can be at most four memory references requesting a bank on the C-240, while there can be at most twelve on the X-MP. However, because of bank conflicts associated with stride-8 references, both SAXPY and V = V + S probes show enormous degradations on the C-240 when the load codes use stride 8.

Because both the probes and loads in these tests are small, tight loops that constantly access memory in a very regular pattern, they are probably worst-case estimates of multi-CPU memory contention effects.

We also measured the times for a full application benchmark when several copies of the same benchmark were running. As shown in Table 4e, HYDRO runs at about 80% efficiency under a four-processor load (efficiency = time with no load / time with multiple copies). On the X-MP/416, the HYDRO efficiency under full load is still about 94%.[16]

5. Software

We would like to make special mention of a software tool we used briefly while at Convex. The tool is SPY and it was written by Ron Gray of Convex, although it is not officially released as a Convex product. SPY is a runtime profiling tool that instruments the code to be profiled at compile time. Following execution, SPY produces a listing that contains, among other things, vector loop lengths (at the code level, not the hardware level), MFLOP rates for each loop, counts for various kinds of instructions, and register-spill information. We are not aware of any products at Los Alamos National Laboratory that provide this information at the loop level. Most interestingly, SPY also works for codes that run in vector concurrent mode, in which case it lists the distribution of logical CPU thread times, showing both wallclock and CPU times. Again, this information is given at the loop level. While we did not have enough time to really make use of SPY to improve our benchmark times, we feel that it would be a welcome tool to any of our existing optimization workbenches. The use of SPY is described more completely in Reference 8.

6. Summary

We found that on our HYDRO benchmark, a code that is representative of a large class of codes in use at the Laboratory, a single C-240 processor ran at about one-third the speed of the CRAY X-MP/14 processor. This suggests that, because the C-240 is a four-processor machine, the C-240 system is about equivalent in throughput power to the CRAY X-MP/14 system. This conclusion takes into account the degradation in runtime we observed while timesharing four HYDRO runs on the four-processor Convex. Of course, the actual "time-to-solution" is the critical factor in many situations, in which case the X-MP/14 has a significant advantage. Also, note that we have chosen the X-MP/14 for comparison because it was the only UNICOS Cray at Los Alamos National Laboratory at the time this study was carried out. However, the X-MP/14 is a member of an older generation of Cray machines; it does not, for example, possess scatter/gather hardware (while the Convex C-240 does). HYDRO runs about 35% faster on a 9.5-ns CRAY X-MP with scatter/gather (under CTSS).

Convex has taken an evolutionary path in the design of its second-generation machine. There was much more of an architectural difference between the CRAY-1 and CRAY X-MP, for example. However, the difference in clock speed between the C-1 and C-240 is much greater than the clock-speed difference between the CRAY-1 and CRAY X-MP. This, of course, is because Crays push the limits of chip and packaging technology and, in doing so, are more limited in the advances they can make in processor speed in any one generation. There is less room to maneuver between four nanoseconds and, say, one nanosecond than there is between forty nanoseconds and one nanosecond. This suggests the minisupercomputers as a group will, for the foreseeable future, continue to make strong gains in clock speed relative to supercomputers.

Acknowledgments

We are grateful to Ron Gray of Convex who has gone well beyond the call of duty in helping us with our benchmarks. Thanks also to George VanDegrift for arranging our Richardson visit.

References

[1] H. J. Wasserman, M. L. Simmons, and O. M. Lubeck, "The Performance of Minisupercomputers: Alliant FX/8, Convex C-1, and SCS-40," *Parallel Comput.* 8 (1988) 285-294.

[2] H. J. Wasserman, "Los Alamos National Laboratory Computer Benchmarking 1988," Los Alamos National Laboratory report LA-11465-MS (December 1988).

[3] M. L. Simmons and H. J. Wasserman, "Los Alamos National Laboratory Computer Benchmarking 1986," Los Alamos National Laboratory report LA-10898-MS (December 1987).

[4] J. H. Griffin and M. L. Simmons, "Los Alamos National Laboratory Computer Benchmarking 1983," Los Alamos National Laboratory report LA-10151-MS (June 1984).

[5] J. L Martin, "Los Alamos National Laboratory Computer Benchmarking 1982," Los Alamos National Laboratory report LA-9698-MS (June 1983).

[6] J. L. Martin and R. G. Martinez, "Los Alamos National Laboratory Computer Benchmarking 1981," Los Alamos National Laboratory report LA-9104-MS (November 1981).

[7] I. Y. Bucher and A. H. Hayes, "Los Alamos National Laboratory Computer Benchmarking 1979," Los Alamos National Laboratory report LA-8689-MS (February 1981).

[8] R. Gray, "Parallelization of a Subset of the Los Alamos Benchmark Suite on the Convex C-240 and Performance Comparisons to the CRAY Series of Computers," Convex Computer Corp., available from the author.

[9] M. Chastain, G. Gostin, J. Mankovich, and S. Wallach, "The Convex C240 Architecture," *Proc. Supercomputing '88*, IEEE Computer Society, 1988, pp. 321-329.

[10] H. J. Wasserman, "Performance of CFT and CFT77 Compilers," Los Alamos National Laboratory internal memorandum BM-89-1206 (December 1989).

[11] R. J. Koskela and H. J. Wasserman, "Benchmark of the Multiflow Trace-300/14," Los Alamos National Laboratory internal memorandum BM-89-1219 (December 1989).

[12] O. Lubeck, J. Moore, and R. Mendez, "A Benchmark Comparison of Three Supercomputers: Fujitsu VP-200, Hitachi S810/20, and Cray X-MP/2," *IEEE Computer Magazine* 12 (1985) 10-24.

[13] M. L. Simmons and H. J. Wasserman, "Performance Comparison of the CRAY-2 and CRAY X-MP/416 Supercomputers," *Proc. Supercomputing '88*, IEEE Computer Society, 1988, pp. 288-295.

[14] H. J. Wasserman and M. L. Simmons, "Benchmark of the CRAY Y-MP/832 SN 1002," Los Alamos National Laboratory internal memorandum C-3-HJW-1 (August 15, 1988).

[15] G. J. Faanes and J. L Schwarzmeier, "Comparing the Performance of CRAY Y-MP and CRAY X-MP Computer Systems," *Cray Channels* 10 (1989) 26-30.